Eigenfaces for Face-Recognition

A new and upcoming face-recognition technique is the topic of discussion here – the 'Eigenfaces-Approach'.
In this, face images are projected onto a feature space that spans significant variations among known face images, locating and tracking a subject's head, and then recognizing the person by comparing characteristics of the face to those of known individuals.

The scheme is based on an information theory approach that decomposes face images into a small set of characteristic feature images called "eigenfaces," which can be considered as the principal components of the initial training set of face images. Recognition is performed by projecting a new image into the subspace spanned by the eigenfaces ("face space") and then classifying the face by comparing its position in face space with the positions of known individuals.

One particular advantage of the Eigenfaces approach is that it provides for the ability to learn and later recognize new faces in an unsupervised manner. Additionally, it is easy to implement using a neural network architecture. These advantages make the Eigenfaces approach a powerful tool for face recognition in practical applications.

The steps included in the process are expanded upon briefly (in a mathematical approach) as follows:

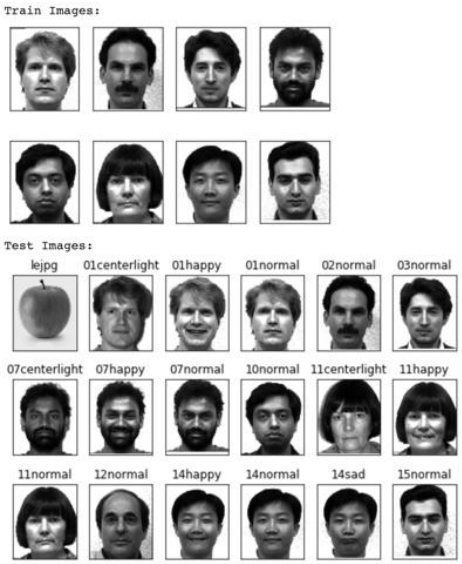
- We obtain M face images as training dataset faces. These images can be understood in simple terms in the forms of matrix.
- Each image (say I_i) is represented as a vector. A mean face vector is calculated.
- The differences between each image's 'information' stored and the average is calculated.
- Then a matrix of these 'difference' vectors is created and a covariance matrix (C) is obtained which is equal to the product of the covariance matrix and its transpose matrix divided by M.
- Then eigenvalues and eigenvectors of C from the covariance matrix C are computed by the computer.
- It's made sure that the eigenvectors are normalized (i.e. the magnitude of the vector has to be 1). These normalized eigenvectors obtained are called as 'eigenfaces'.
- Then images are reconstructed by adding the mean face vector to the product of the eigenfaces and the weight contribution of each faces. These reconstructed images are compared with vectors of original images and the difference between them is termed as root mean square error.
- Now the same process takes place with a new 'test' image, and so the weights of the test image are computed.
- Then the difference between the weight-matrices (the matrices containing information about the weight contribution) of the new test image and the original face class is found out. This is termed as the Euclidean distance, e_d.
- When minimum e_d is less than a Threshold (T) amount, then the new/test face is classified as belonging to the original class; otherwise the test face is classified as unknown. The Threshold (T) amount is usually taken as half the largest distance between any two face images in the training set.

A python code to calculate and implement the eigenface technique is explained below:
[Note: The dataset implemented in the code consists of a set of 17 grayscale face images (in JPG format). Each image is of dimension 195 x 231 (width x height) pixels and each pixel uses 8 bits for grayscale. The following 8 images are used as training images: subject01.normal, subject02.normal, subject03.normal, subject07.normal, subject10.normal, subject11.normal, subject14.normal and subject15.normal. All 17 images (including the 8 training images) in the dataset are used as test images. A non-face image (apple1) in JPG format is also attached. This image is also of dimension 195 x 231. The code is also tested on this non-face image to check the validity.**]**

```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
import os
```

Here, the basic libraries are imported.

```
dataset_path = 'Dataset/'
dataset_dir  = os.listdir(dataset_path)
width  = 195
height = 231
print('Train Images:')
train_image_names = ['subject01.normal.jpg', 'subject02.normal.jpg', 'subject03.normal.jpg', 'subject07.normal.jpg', 'subject10.normal.jpg', 'subject11.normal.jpg', 'subject14.normal.jpg', 'subject15.normal.jpg']
training_tensor   = np.ndarray(shape=(len(train_image_names), height*width), dtype=np.float64)
for i in range(len(train_image_names)):
    img = plt.imread(dataset_path + train_image_names[i])
    training_tensor[i,:] = np.array(img, dtype='float64').flatten()
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='gray')
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
print('Test Images:')
test_image_names = dataset_dir
testing_tensor   = np.ndarray(shape=(len(test_image_names), height*width), dtype=np.float64)
for i in range(len(test_image_names)):
    img = imread(dataset_path + test_image_names[i])
    testing_tensor[i,:] = np.array(img, dtype='float64').flatten()
    plt.subplot(3,6,1+i)
    plt.title(test_image_names[i].split('.')[0][-2:]+test_image_names[i].split('.')[1])
    plt.imshow(img, cmap='gray')
    plt.subplots_adjust(right=1.2, top=1.2)
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
mean_face = np.zeros((1,height*width))
for i in training_tensor:
    mean_face = np.add(mean_face,i)
```



The test images are read.

```
mean_face = np.divide(mean_face,float(len(train_image_names))).flatten()
plt.imshow(mean_face.reshape(height, width), cmap='gray')
plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
```

The mean face is formulated.

```
normalised_training_tensor = np.ndarray(shape=(len(train_image_names), height*width))
for i in range(len(train_image_names)):
    normalised_training_tensor[i] = np.subtract(training_tensor[i],mean_face)
for i in range(len(train_image_names)):
    img = normalised_training_tensor[i].reshape(height,width)
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='gray')
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
```

Normalized faces are computed and displayed.

```
cov_matrix = np.cov(normalised_training_tensor)
cov_matrix = np.divide(cov_matrix,8.0)
print('Covariance matrix of X: \n%s' %cov_matrix)
eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix)
print('Eigenvectors of Cov(X): \n%s' %eigenvectors)
print('\nEigenvalues of Cov(X): \n%s' %eigenvalues)
eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]
eig_pairs.sort(reverse=True)
eigvalues_sort  = [eig_pairs[index][0] for index in range(len(eigenvalues))]
eigvectors_sort = [eig_pairs[index][1] for index in range(len(eigenvalues))]
```

Covariance matrix formulated, and corresponding eigenvectors and eigenvalues are obtained. Example of output displayed for the considered set of data:

Covariance matrix of X:
[[240.21425354 -54.37445049 -49.91300972 -167.04449305   6.71011608
    95.13549119   51.86167951 -122.58958706]
 [-54.37445049 271.21637481 -39.69264581   66.46457924 -62.78262301
  -159.33970561 -96.7188796   75.22735047]
 [-49.91300972 -39.69264581 223.857185   46.24225037 -32.6657127
  -134.1771553   -35.12864547   21.47773363]
 [-167.04449305   66.46457924   46.24225037 345.77440281 -80.00529939
  -267.06328206 -73.29909108 128.93093316]
 [6.71011608 -62.78262301 -32.6657127   -80.00529939 256.35235515

-63.58037876 53.02911913 -77.05757652]
[95.13549119 -159.33970561 -134.1771553 -267.06328206 -63.58037876
  747.14220277   1.26270451 -219.37987674]
[51.86167951 -96.7188796   -35.12864547 -73.29909108   53.02911913
    1.26270451 238.6557604 -139.66264741]
[-122.58958706   75.22735047   21.47773363 128.93093316 -77.05757652
 -219.37987674 -139.66264741 333.05367046]]

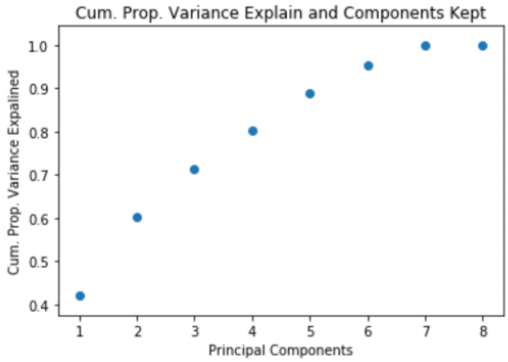
Eigenvectors of Cov(X):
[[0.24216786 -0.23304127 0.35355339 -0.54650028 0.34900164 0.24263134
  -0.46008264 -0.26652539]
 [-0.2320557   0.21004915 0.35355339 0.28336041 0.66156373 -0.40750816
   0.14797794 -0.26666354]
 [-0.15104702 -0.0756895   0.35355339 0.19555006 -0.49779746 -0.46987952
  -0.57605919 -0.08822017]
 [-0.41563654 0.18098142 0.35355339 -0.49133856 -0.36828479 0.0496277
   0.45683638 -0.28510056]
 [0.05114387 -0.48868195 0.35355339 -0.17720876 0.0789403 -0.29622126
   0.28068609 0.65529232]
 [0.73015769 0.53226997 0.35355339 0.03110374 -0.15710617 -0.06932017
   0.14579398 0.08274941]
 [0.14350419 -0.47101419 0.35355339 0.52641965 -0.14266291 0.41459652
   0.25325285 -0.31472933]
 [-0.36823436 0.34512637 0.35355339 0.17861375 0.07634566 0.53607356
  -0.2484054   0.48319727]]

Eigenvalues of Cov(X):
[1.11554951e+03   4.85892625e+02 -8.47167977e-14   1.22955135e+02
   2.95284929e+02   1.76427142e+02   2.32097712e+02   2.28059153e+02]

```
var_comp_sum = np.cumsum(eigvalues_sort)/sum(eigvalues_sort)
print("Cumulative proportion of variance explained vector: \n%s" %var_comp_sum)
num_comp = range(1,len(eigvalues_sort)+1)
plt.title('Cum. Prop. Variance Explain and Components Kept')
plt.xlabel('Principal Components')
plt.ylabel('Cum. Prop. Variance Expalined')
plt.scatter(num_comp, var_comp_sum)
plt.show()
```

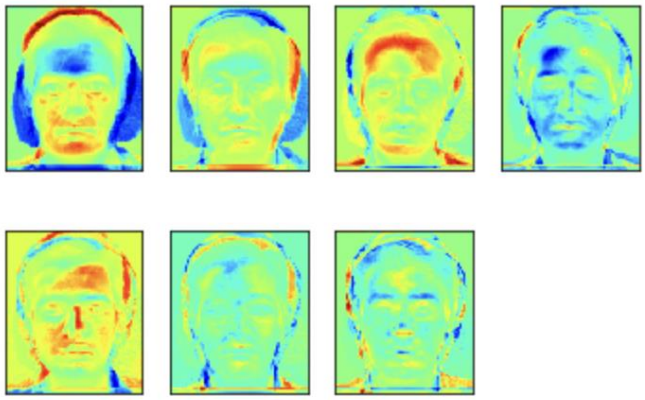
Then a graph of cumulative proportion of variance with respect to the principal components is generated for the sake of a thorough understanding.
Again, continuing with the same considered data, we get output:

Cumulative proportion of variance explained vector:
[0.41996902 0.60289218 0.7140576   0.80143503 0.88729207 0.95371129
  1.   1.   ]




```
reduced_data = np.array(eigvectors_sort[:7]).transpose()
proj_data = np.dot(training_tensor.transpose(),reduced_data)
proj_data = proj_data.transpose()
for i in range(proj_data.shape[0]):
    img = proj_data[i].reshape(height,width)
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='jet')
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
```

Necessary no. of principle components chosen and eigen faces plotted from the projected data.



```
w = np.array([np.dot(proj_data,i) for i in normalised_training_tensor])
w
```

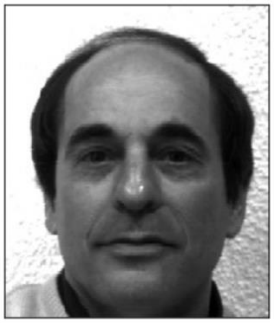
Now, weights are found for each training image, and displayed in an array (a dataset). Continuing with the same data:

```
array([[  8.48287612e+07,  -5.81765456e+07,   4.32101930e+07,
         -7.76647463e+07,  -3.06341089e+07,   3.58272786e+07,
         -3.53176153e+07],
       [ -8.90553605e+07,   4.26457404e+07,   6.83431225e+07,
          2.56112699e+07,  -1.89660204e+07,  -3.27984752e+07,
          1.63052275e+07],
       [ -6.49888673e+07,  -1.91766896e+07,  -5.08976401e+07,
         -6.15416348e+07,  -1.02271740e+07,  -2.29160969e+07,
          4.87793668e+06],
       [ -1.63772788e+08,   3.62801041e+07,  -4.07933003e+07,
```

```
  4.85649446e+07,  -2.11225024e+07,  -2.23707416e+06,
 -1.88350738e+07],
[  2.50823074e+07,  -7.92886248e+07,   6.20549912e+06,
   3.76311039e+07,   5.70068630e+07,  -2.62025963e+07,
  -3.84041861e+06],
[  3.03507468e+08,   1.07059520e+08,  -2.15641534e+07,
   4.34624113e+07,   1.37673278e+07,  -2.06872576e+07,
   1.02386688e+07],
[  5.73401449e+07,  -8.29527225e+07,  -1.50118335e+07,
   2.00947251e+07,  -2.61070419e+07,   2.69240539e+07,
   2.30162486e+07],
[ -1.52941665e+08,   5.36092179e+07,   1.05081126e+07,
  -3.61580737e+07,   3.62826568e+07,   4.20901677e+07,
   3.55502602e+06]])
```

Now, we come to the recognition and the subsequent processing of the unknown face. Like the previous steps, it is normalized and the weights are formulated.

```
unknown_face        = plt.imread('Dataset/subject12.normal.jpg')
unknown_face_vector = np.array(unknown_face, dtype='float64').flatten()
plt.imshow(unknown_face, cmap='gray')
plt.title('Unknown face')
plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
normalised_uface_vector = np.subtract(unknown_face_vector,mean_face)
plt.imshow(normalised_uface_vector.reshape(height, width), cmap='gray')
plt.title('Normalised unknown face')
plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
plt.show()
w_unknown = np.dot(proj_data, unknown_face_vector)
w_unknown
```

Unknown face  ➔  Normalized unknown face

Now the difference between the weights of the unknown face and the weights of the original data class is calculated.

```
diff  = w - w_unknown
norms = np.linalg.norm(diff, axis=1)
print(norms)
min(norms)
```

Then the faces are processed and checked if they are recognized.

```
count        = 0
num_images   = 0
correct_pred = 0
def recogniser(img, train_image_names,proj_data,w):
    global count,highest_min,num_images,correct_pred
    unknown_face        = plt.imread('Dataset/'+img)
    num_images          += 1
    unknown_face_vector = np.array(unknown_face, dtype='float64').flatten()
    normalised_uface_vector = np.subtract(unknown_face_vector,mean_face)
    plt.subplot(9,4,1+count)
    plt.imshow(unknown_face, cmap='gray')
    plt.title('Input:'+'.'.join(img.split('.')[:2]))
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    count+=1
    w_unknown = np.dot(proj_data, normalised_uface_vector)
    diff  = w - w_unknown
    norms = np.linalg.norm(diff, axis=1)
    index = np.argmin(norms)
    t1 = 100111536
    t0 = 88831687
```

```
    if norms[index] < t1:
        plt.subplot(9,4,1+count)
        if norms[index] < t0:
            if img.split('.')[0] == train_image_names[index].split('.')[0]:
                plt.title('Matched:'+'.'.join(train_image_names[index].split('.')[:2]), color='g')
                plt.imshow(imread('Dataset/'+train_image_names[index]), cmap='gray')
                correct_pred += 1
            else:
                plt.title('Matched:'+'.'.join(train_image_names[index].split('.')[:2]), color='r')
                plt.imshow(imread('Dataset/'+train_image_names[index]), cmap='gray')
        else:
            if img.split('.')[0] not in [i.split('.')[0] for i in train_image_names] and img.split('.')[0] != 'apple':
                plt.title('Unknown face!', color='g')
                correct_pred += 1
            else:
                plt.title('Unknown face!', color='r')
        plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
        plt.subplots_adjust(right=1.2, top=2.5)
    else:
        plt.subplot(9,4,1+count)
        if len(img.split('.')) == 3:
            plt.title('Not a face!', color='r')
        else:
            plt.title('Not a face!', color='g')
            correct_pred += 1
        plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    count+=1
```

```
fig = plt.figure(figsize=(15, 15))
for i in range(len(test_image_names)):
    recogniser(test_image_names[i], train_image_names,proj_data,w)
plt.show()
print('Correct predictions: {}/{} = {}%'.format(correct_pred, num_images, correct_pred/num_images*100.00))
count = 0
def recogniser(img, train_image_names,proj_data,w):
    global count
    unknown_face        = plt.imread('Dataset/'+img)
    unknown_face_vector = np.array(unknown_face, dtype='float64').flatten()
    normalised_uface_vector = np.subtract(unknown_face_vector,mean_face)
    plt.subplot(9,4,1+count)
    plt.imshow(unknown_face, cmap='gray')
    plt.title('Input:'+'.'.join(img.split('.')[:2]))
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    count+=1
    plt.subplot(9,4,1+count)
    plt.imshow(normalised_uface_vector.reshape(height, width), cmap='gray')
    plt.title('Normalised Face')
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    plt.subplots_adjust(right=1.2, top=2.5)
    count+=1
fig = plt.figure(figsize=(15, 15))
for i in range(len(test_image_names)):
    recogniser(test_image_names[i], train_image_names,proj_data,w)
plt.show()
```

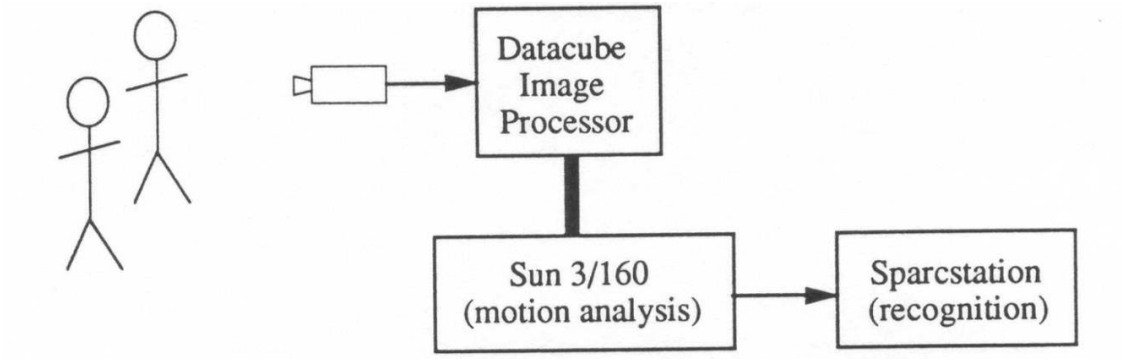
Output from the set of data considered:
Correct predictions: 14/18 = 77.77777777777779%

*****

The eigenface approach poses some issues in the functioning as well, which are mentioned below along with a few possible solutions:

- The Background Problem: The background of an image can significantly affect facial recognition performance because the eigenface analysis does not distinguish the face from the rest of the image. To deal with this problem, the authors suggest multiplying the input face image by a two-dimensional Gaussian window centred on the face. This technique emphasizes the middle of the face and diminishes the background, without requiring the difficult task of robust segmentation of the head. Additionally, de-emphasizing the outside of the face is a practical consideration as changing hairstyles may negatively affect recognition.
- Scale and Orientation Invariance: Experiments conducted show that recognition performance is affected by misjudged head size. It can be solved by estimating using motion analysis. Multiscale eigenfaces and scaling the input image to multiple sizes can also be used to address the scale problem.
- Multiple Views: The approach, till now, works only with full frontal view images, and constant work is being done to improve upon this. For example: a system defining a limited number of face classes for each known person corresponding to characteristic views such as frontal, side, 45 degrees, and right and left profile views has been proposed and worked upon.
- A Problem in the Face Space: The nearest-neighbour classification assumes a Gaussian distribution in face space of an individual's feature vectors. Since there is no a priori reason to assume any particular distribution, characterizing it is better rather than assuming it to be gaussian. Using nonlinear networks can serve as a promising way to learn the face space distributions.

Few developments on this topic have shown progress as well. A system has been built that is capable of locating and recognizing faces in real-time using a fixed camera connected to a Datacube image processing system and a Sun 3/160 computer for motion detection and analysis. The system performs spatiotemporal filtering, thresholding, and sub-sampling to detect moving objects and tracks motion to determine if it is tracking a head. When a head is found, the subimage is sent to a Sun Sparcstation computer running the face recognition program. Recognition occurs at rates of up to two or three times per second. When a face is recognized, the image of the identified individual is displayed on the Sun monitor. The system uses a distance-from-face-space measure to recognize faces and can either reject them as not a face, recognize them as one of a group of familiar faces, or determine them to be an unknown face.



There have been biological motivations for face recognition also. Potential has been shown for a recognition mechanism based on fast, low-level, two-dimensional image processing. The existence of such a mechanism is supported by physiological experiments in monkey cortex, which have claimed to isolate neurons that respond selectively to faces.

This has also been concluded from studies that the eigenfaces-approach to face recognition has qualitative similarities with human face recognition. For instance, relatively small changes cause the recognition to degrade gracefully, so that partially occluded faces can be recognized. Gradual changes due to aging are easily handled by the occasional recalculation of the eigenfaces, so that the system is quite tolerant to even large changes as long as they occur over a long period of time. If, however, a large change occurs quickly e.g., addition of a disguise or change of facial hair, then the eigenfaces approach will be fooled, as are people in conditions of casual observation.

The eigenface recognition system may be implemented using simple parallel computing elements, as in a connectionist system or artificial neural network. There's an established three-layer neural network that implements a significant part of the eigenfaces system, with the input layer receiving the input face image and the weights from the input layer to the hidden layer corresponding to the eigenfaces. The output layer produces the face space projection of the input image, and the classification vector reveals the identity of the input face image. The network has similarities to associative networks and can be used to recall partially occluded faces.

Conclusions

The eigenface approach to face recognition proves to be a successful method of face recognition which can have various practical applications. Although not the most elegant, it is fast, relatively simple, and has been shown to work well in a constrained environment. user. For applications such as security systems or human-computer interaction, the system will normally be able to "view" the subject for a few seconds or minutes, and thus will have a number of chances to recognize the person. Experiments show that the eigenface technique can be made to perform at very high accuracy and thus is well suited to these types of applications.
Researchers have been investigating the issues of robustness to changes in lighting, head size, and head orientation, automatically learning new faces, incorporating a limited number of characteristic views for each individual, and the trade-offs between the number of people the system needs to recognize and the number of eigenfaces necessary for unambiguous

classification. The researchers plan to improve upon these and make the eigenface technique the best it can be. They want to improve by introducing new aspects too, such as-using eigenface analysis to determine the gender of the subject and interpret facial expressions.

*****

Thank you.
By - Krishang Sharma